Booting Process

22 Nov 2024

OUTLINE

1. Reset Source
2. Booting Process
3. Bootloader
4. StartUp Code

1. Reset Source

Types of reset:

1. System reset
2. Power reset
3. Backup domain reset

1. Reset Source

1. System reset:

- ❑ Resets all registers except the RCC registers and the backup domain
- ❑ Sources
  - Low level on the NRST pin (External Reset)
  - WWDG end of count condition (WWDG reset)
  - IWDG end of count condition (IWDG reset)
  - A software reset (through NVIC)
  - Low power management reset (Standby/Stop entry)

1. Reset Source

2. Power reset:

- ❑ Resets all registers except the backup domain
- ❑ Sources
  - Power-on/Power-down reset (POR/PDR)
  - Brownout (BOR) reset
  - When exiting the Standby mode

1. Reset Source

3. Backup domain reset:

- ❑ Resets in the backup domain: RTC + Backup + RCC BDCR register
- ❑ Sources
  - BDRST bit in RCC BDCR register set by software
  - VBAT and VDD power-on

1. Reset Source

RCC clock control & status register (RCC_CSR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LPWR RSTF	WWDG RSTF	IWDG RSTF	SFT RSTF	POR RSTF	PIN RSTF	BORRS TF	RMVF	Reserved							
r	r	r	r	r	r	r	rt_w								
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved														LSIRDY	LSION
														r	rw

Bit 31 LPWRRSTF: Low-power reset flag
Bit 30 WWDGRSTF: Window watchdog reset flag
Bit 29 IWDGRSTF: Independent watchdog reset flag
Bit 28 SFTRSTF: Software reset flag
Bit 27 PORRSTF: POR/PDR reset flag
Bit 26 PINRSTF: PIN reset flag
Bit 25 BORRSTF: BOR reset flag

2. Booting Process

1. Vector table:

- The vector table contains the initialization value for the stack pointer, and the entry point addresses of each exception handler.
- The vector table is normally defined in the startup codes provided by the microcontroller vendors.
- By default, the vector table starts at memory address 0 and it will usually be either flash memory or ROM devices.
- However, in some applications can re-allocated vector table at run time by modify the Vector Table Offset Register (VTOR) in System control block(SCB)

Memory Address			Exception Number
		1	
		1	
0x0000004C	Interrupt#3 vector	1	19
0x00000048	Interrupt#2 vector	1	18
0x00000044	Interrupt#1 vector	1	17
0x00000040	Interrupt#0 vector	1	16
0x0000003C	SysTick vector	1	15
0x00000038	PendSV vector	1	14
0x00000034	Not used		13
0x00000030	Debug Monitor vector	1	12
0x0000002C	SVC vector	1	11
0x00000028	Not used		10
0x00000024	Not used		9
0x00000020	Not used		8
0x0000001C	Not used		7
0x00000018	Usage Fault vector	1	6
0x00000014	Bus Fault vector	1	5
0x00000010	MemManage vector	1	4
0x0000000C	HardFault vector	1	3
0x00000008	NMI vector	1	2
0x00000004	Reset vector	1	1
0x00000000	MSP initial value		0

2. Booting Process

2. Booting process:

- Power supply(power on - PON)
- Hardware initialization
- Select boot region??
- Init stack pointer
- Execute Reset_Handler()
- User initialization
- Execute main()

Memory Address			Exception Number
		1	
		1	
0x0000004C	Interrupt#3 vector	1	19
0x00000048	Interrupt#2 vector	1	18
0x00000044	Interrupt#1 vector	1	17
0x00000040	Interrupt#0 vector	1	16
0x0000003C	SysTick vector	1	15
0x00000038	PendSV vector	1	14
0x00000034	Not used		13
0x00000030	Debug Monitor vector	1	12
0x0000002C	SVC vector	1	11
0x00000028	Not used		10
0x00000024	Not used		9
0x00000020	Not used		8
0x0000001C	Not used		7
0x00000018	Usage Fault vector	1	6
0x00000014	Bus Fault vector	1	5
0x00000010	MemManage vector	1	4
0x0000000C	HardFault vector	1	3
0x00000008	NMI vector	1	2
0x00000004	Reset vector	1	1
0x00000000	MSP initial value		0

Value to initialize the Program Counter (PC)

Value to initialize the Main Stack Pointer (MSP)

2. Booting Process

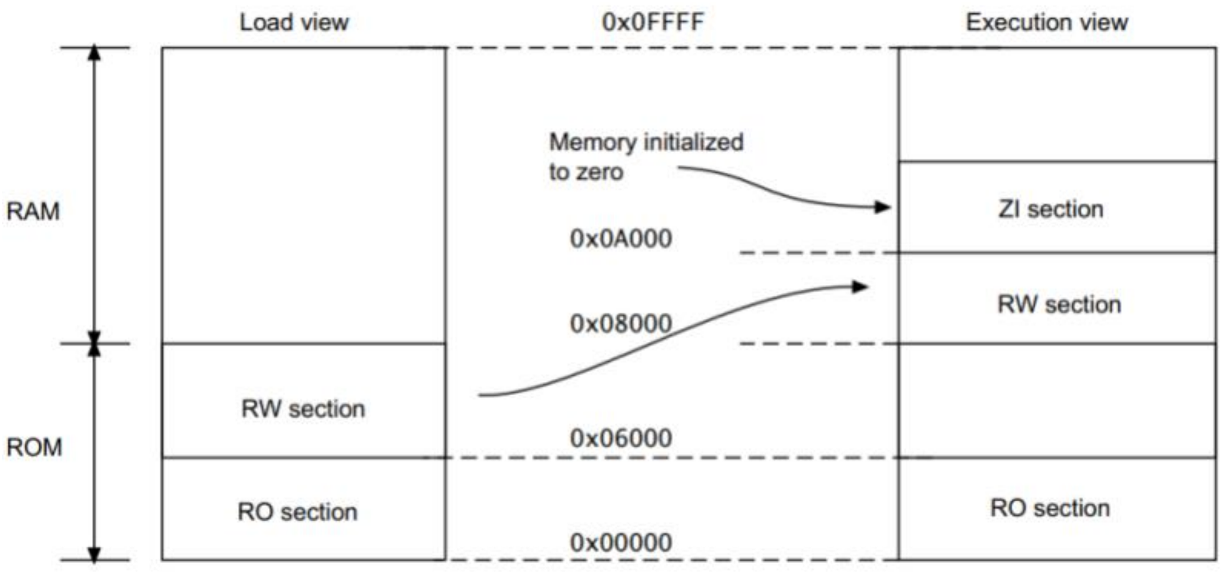
2. Booting process:

- PC is load with the value 0x00000000.
- The processor read the value at 0x00000000 and load this value into Main stack pointer (MSP).
- The processor read the value at 0x00000004 and load this value into PC (program counter), this value is actually address of reset handler.
- PC jump to reset handler function. Reset handler is just a C or assembly function written by user to carry out any initialization required.
- From reset handler, user call main function of the application.

2. Booting Process

3. Reset_handler()

1. Disable all interrupts: All Interrupts Service will not be executed(E.g. Watchdog…)
2. Initialize .data segment: Move data from data segment from ROM to RAM.
3. Initialize the .bss segment: Reset value in bss section to zero.
4. Initialize stack segment: Reserve stack memory in RAM base on linker file.
5. Enable interrupts
6. Call main function



2. Booting Process

4. Boot configuration

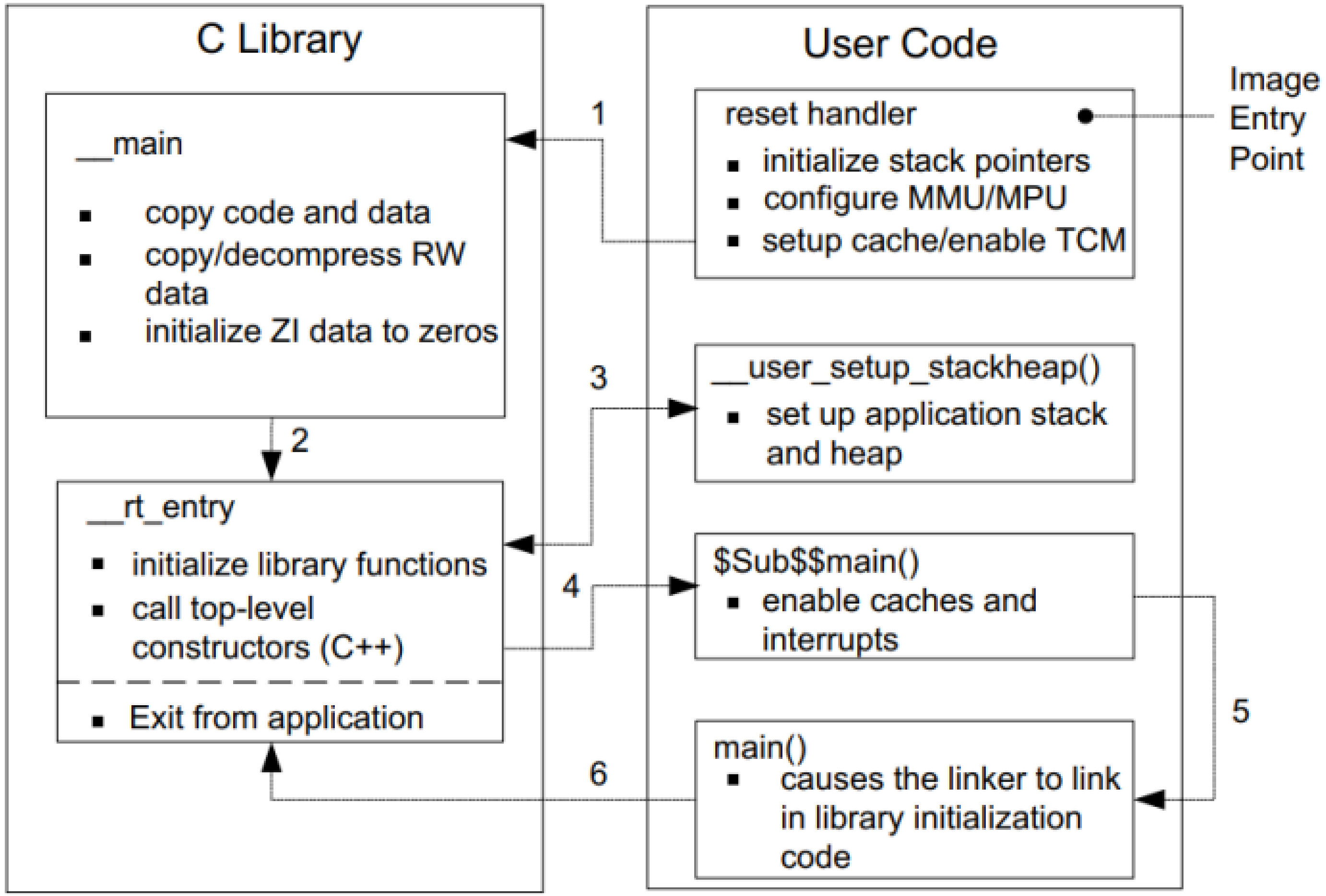
Table 2. Boot modes

Boot mode selection pins		Boot mode	Aliasing
BOOT1	BOOT0		
x	0	Main Flash memory	Main Flash memory is selected as the boot space
0	1	System memory	System memory is selected as the boot space
1	1	Embedded SRAM	Embedded SRAM is selected as the boot space

Block	Addresses	Size
Main Flash memory	0x0800 0000 - 0x081F FFFF	2 Mbytes
System memory	0x1FFF 0000 - 0x1FFF 77FF	30 Kbytes
Embedded SRAM	0x2000 0000 - 0x2004 0000	256 Kbytes

5. Initialization Sequence:

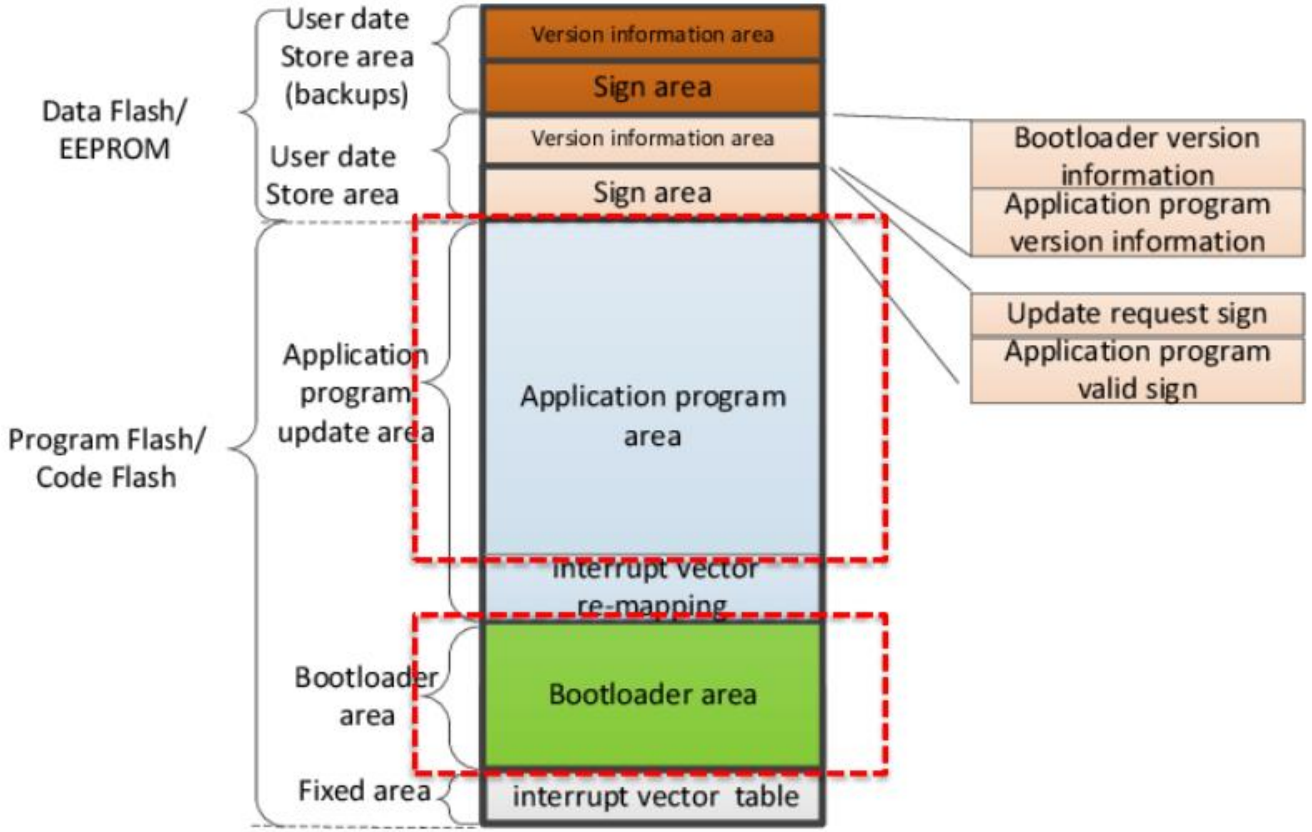
The following figure shows a possible initialization sequence for an embedded system based on an Arm architecture:



3. Bootloader

1. What is Bootloader?

- The bootloader is a program that boots the system or operating system that has been programmed and stored in the device's ROM/Flash
- The bootloader is executed before the application.

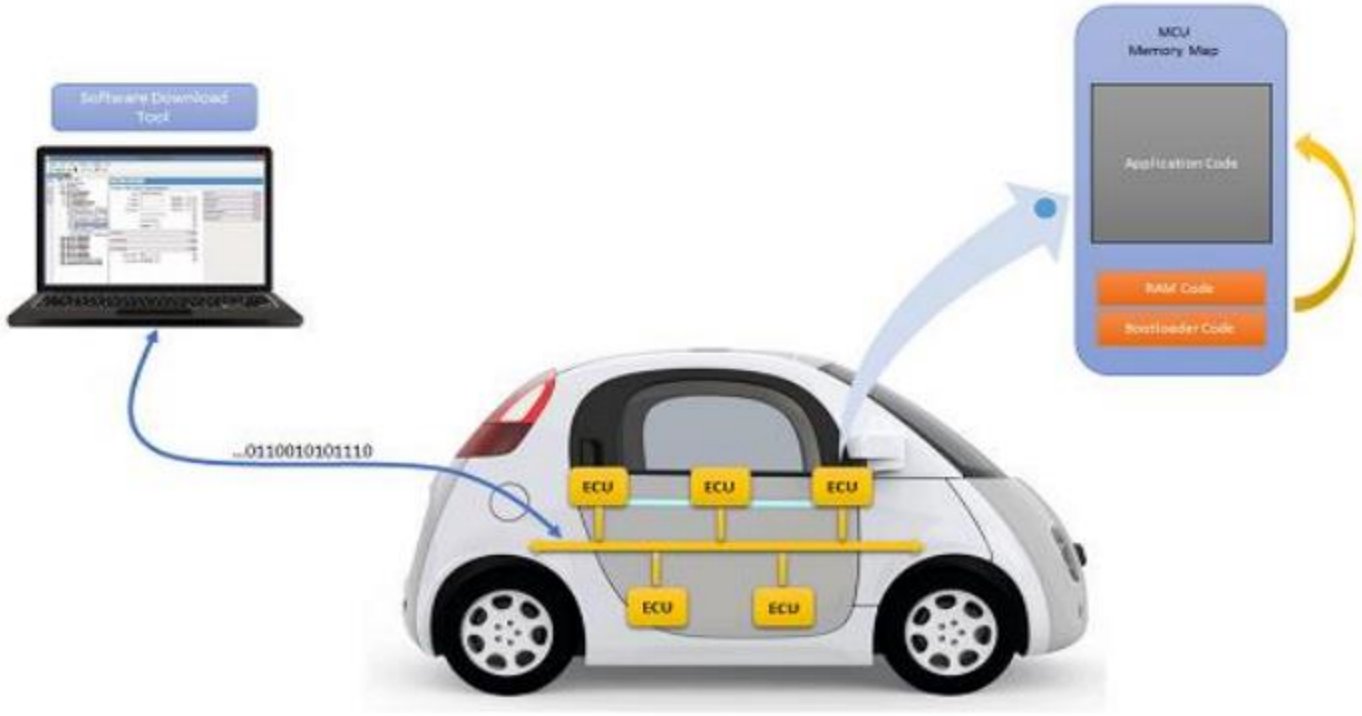


3. Bootloader

2. What is the purpose of Bootloader?

❑ Use to update the Application:

- After the application has been corrected.
- After the application is added the new feature.
- After the customer changes the request (Requirement)…



3. Bootloader

3. The protocols that Bootloader uses:

- CAN
- LIN
- FlexRay
- Ethernet
- OTA
- …

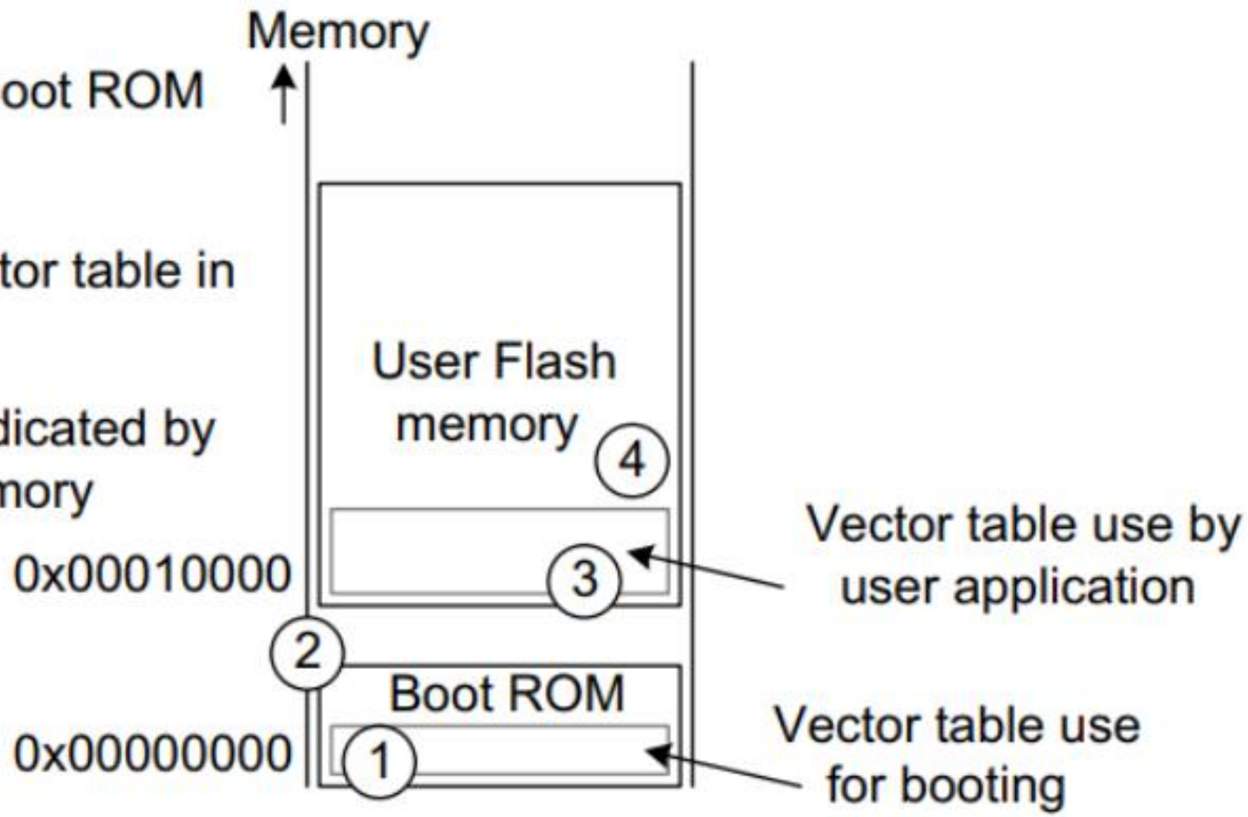
4. The switch mechanism from bootloader to application:

1) Boot up using vector table in boot ROM

2) Carry out boot loader tasks

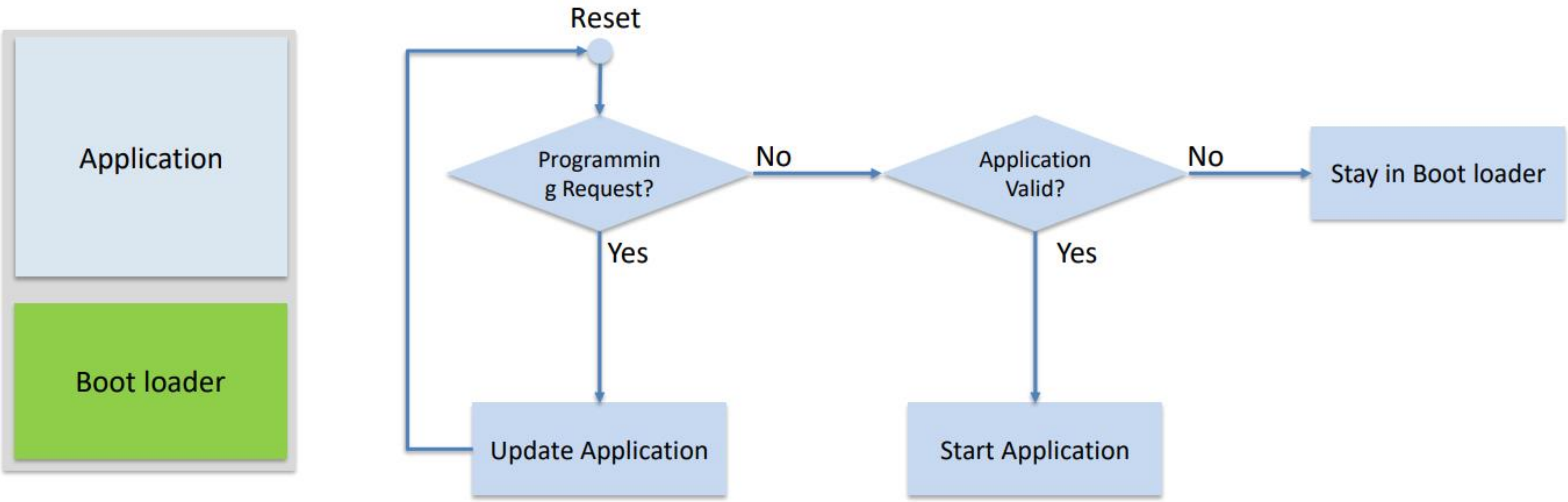
3) Program VTOR to point to vector table in user flash

4) Branch to the reset handler indicated by the vector table of user flash memory



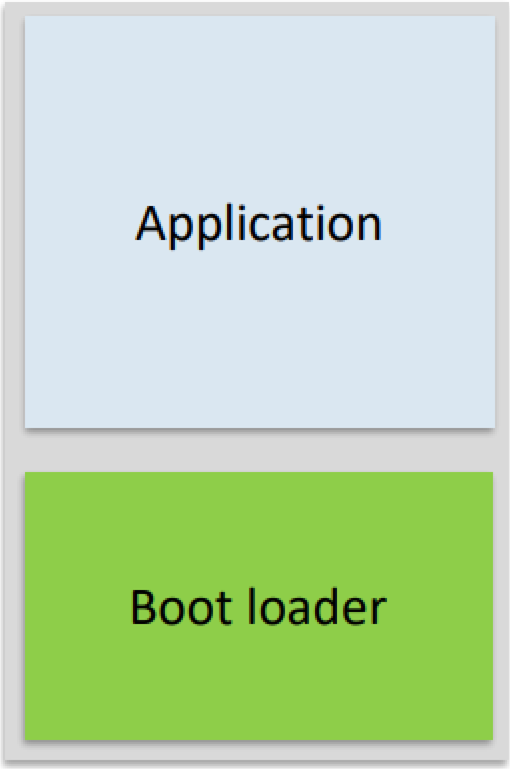
3. Bootloader

5. Boot Loader's activity in the simplest case : 1 Bootloader and 1 Application.

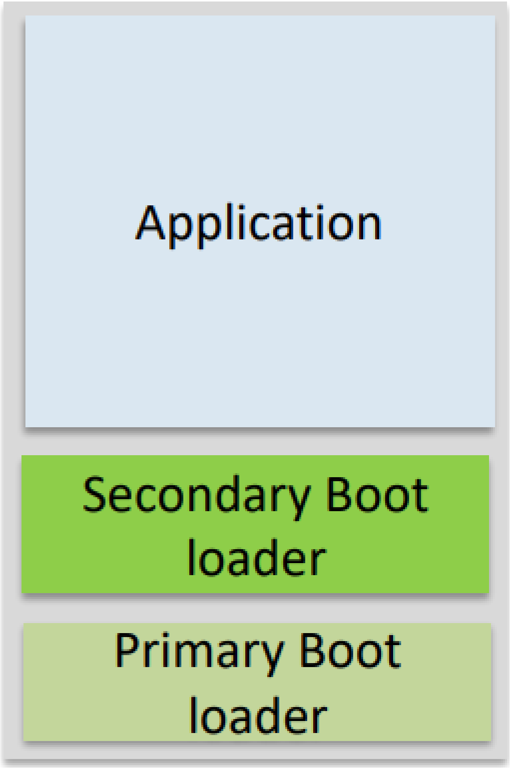


3. Bootloader

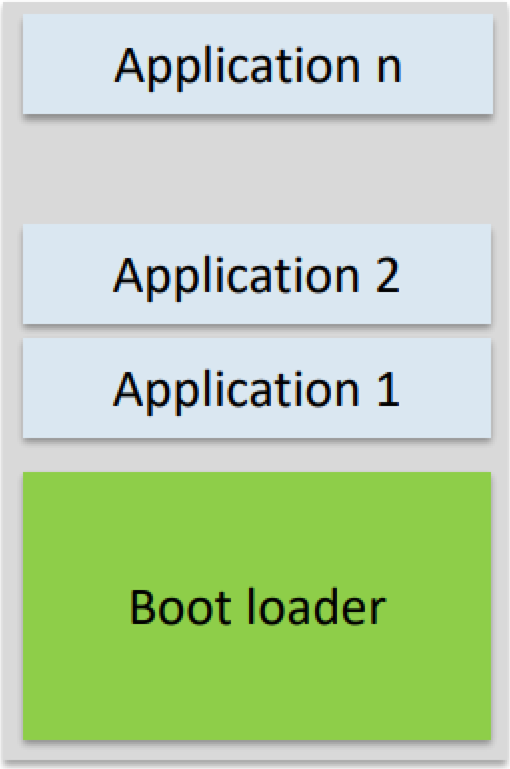
6. Types of Bootloader:



1 boot loader
1 application



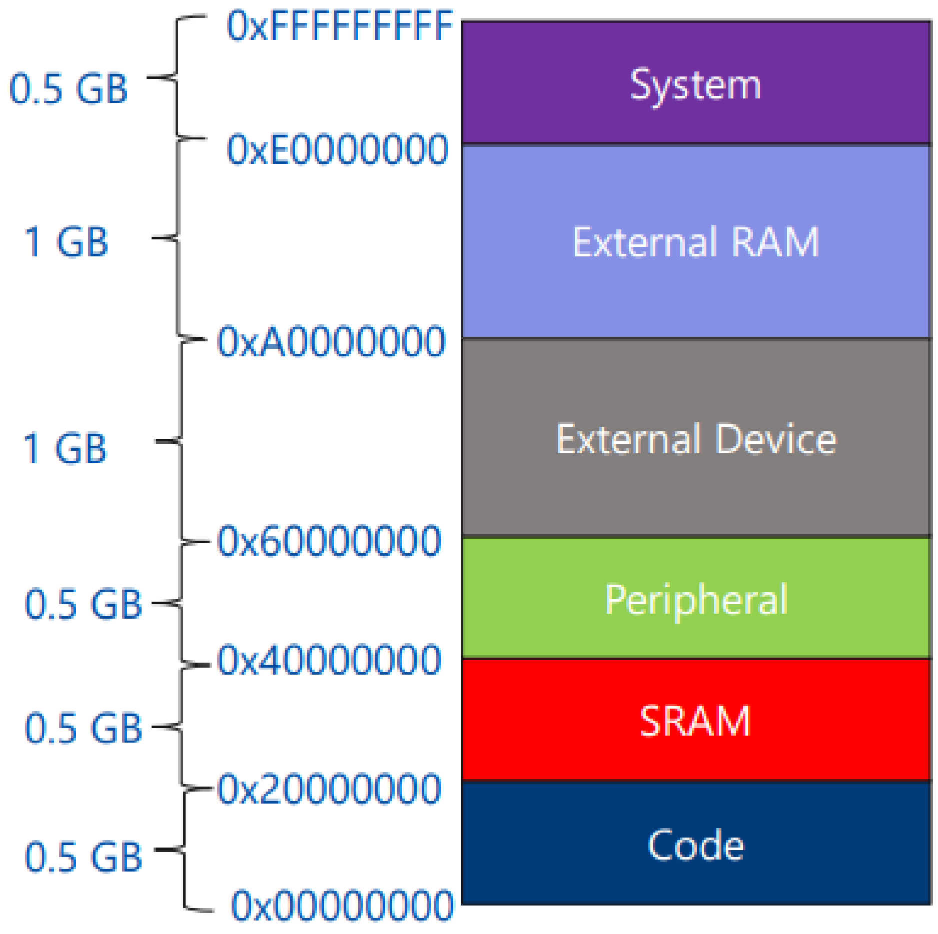
2 boot loader
1 application

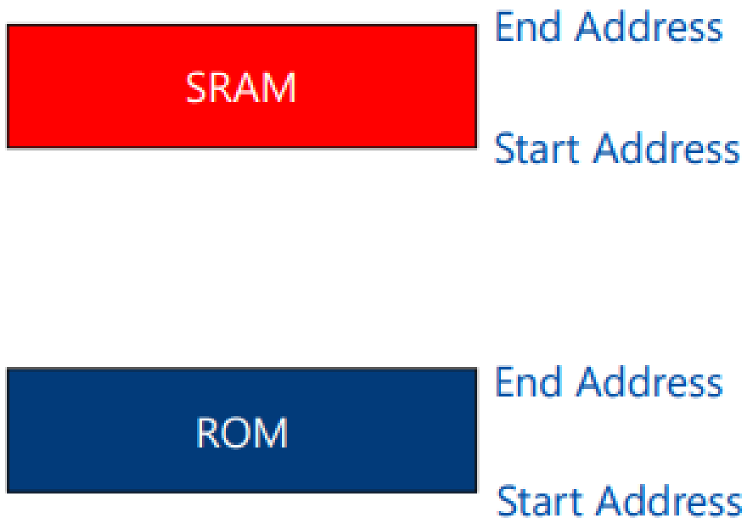


1 boot loader
n applications

4. StartUp Code

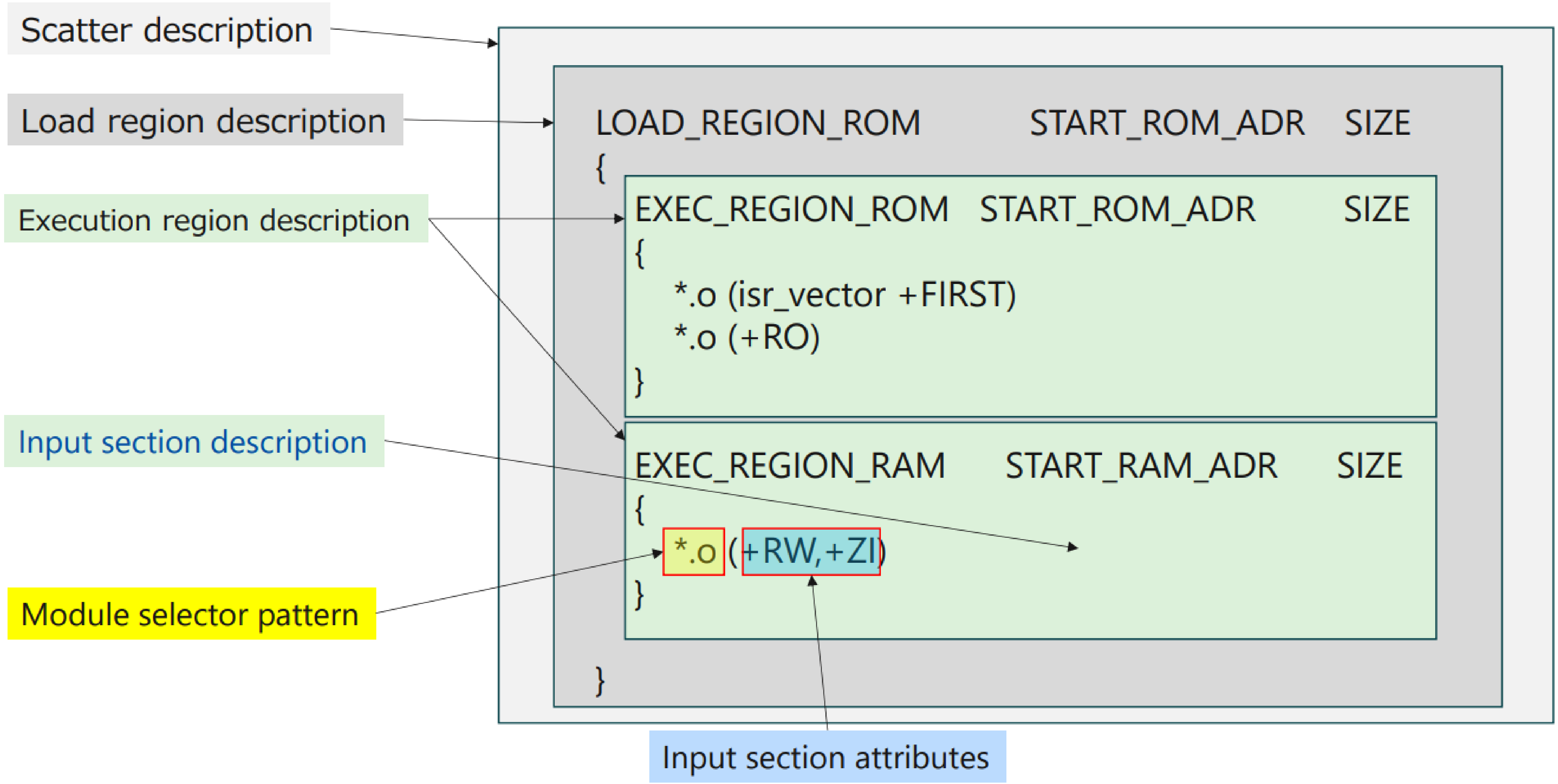
1. Memory





4. StartUp Code

2. Linker Script

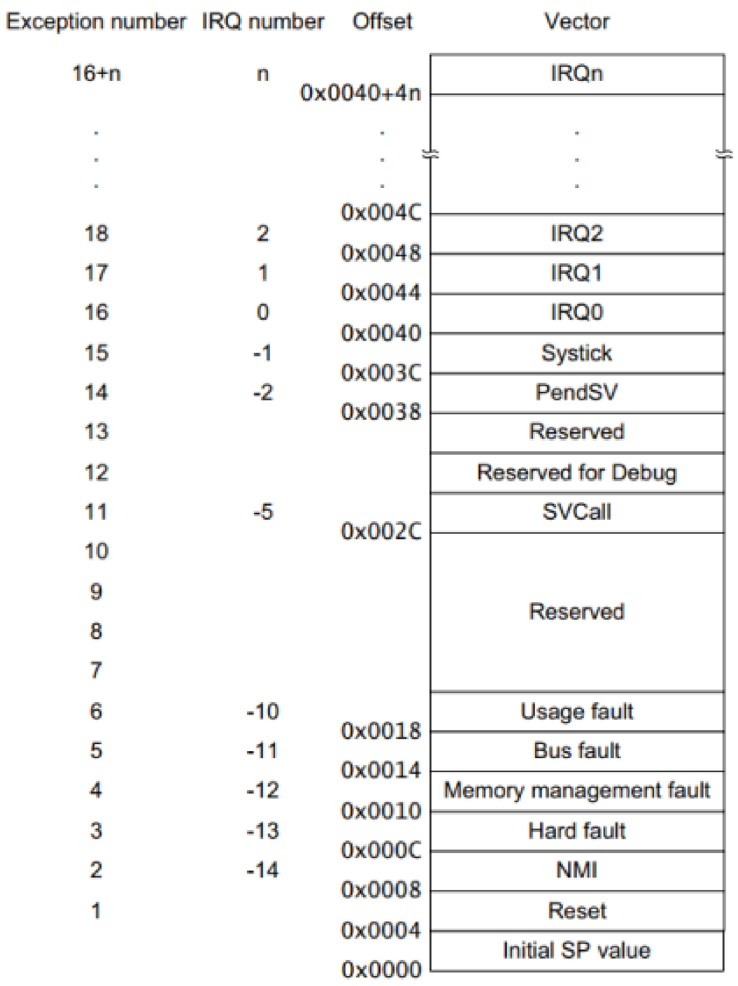


4. StartUp Code

3. Vector table

To create a vector table for interrupts using an array and map it into a specific section (isr_vector):

```
static const unsigned int vector_table[]
__attribute__( (used, section("isr_vector") )) =
{
    (unsigned int)0x20004000,
    (unsigned int)&Reset_Handler
};
```



Vector Table Array: The vector_table array is defined to hold pointers to the ISRs. The first entry is the initial stack pointer, followed by the address of each ISR.

4. StartUp Code

4. Mapping Code and Data to the Target

C1: Using __attribute__

__attribute__((section("name")))

C2: Using #pragma clang section

#pragma clang section [section_type_list]

4. StartUp Code

5. Compiler-specific

Function attributes:

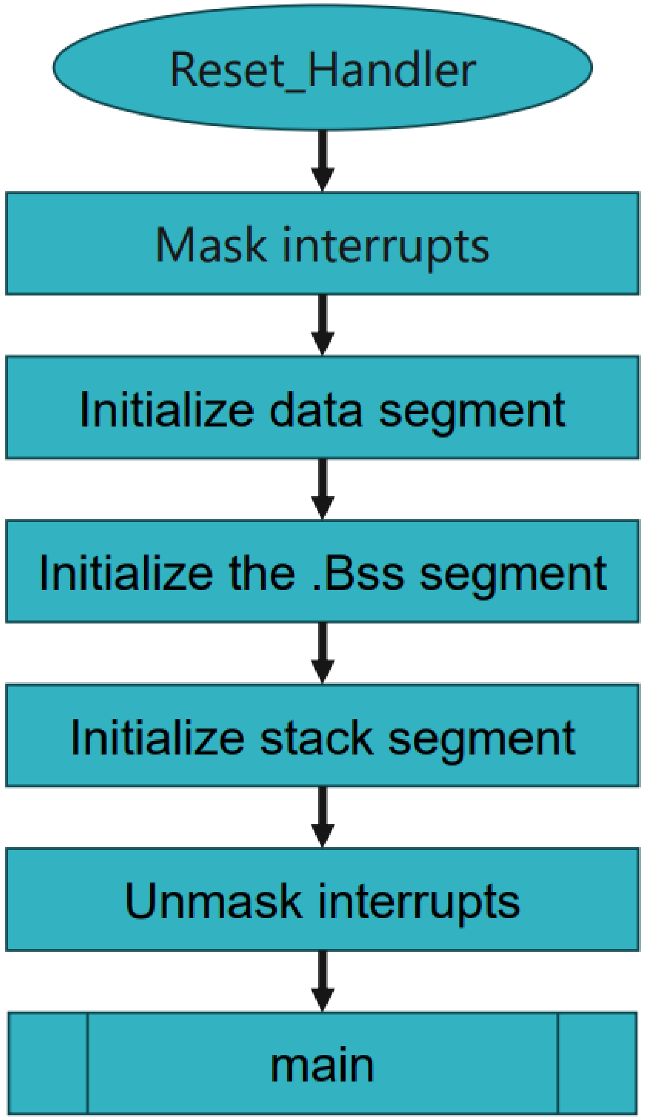
```
__attribute__((alias))
__attribute__((section("name")))
__attribute__((used))
__attribute__((weak))
```

Variable attributes:

```
__attribute__((section("name")))
__attribute__((used))
```

4. StartUp Code

6. Reset_Handler



4. StartUp Code

7. Inline assembly code

The compiler provides an inline assembler that enables you to write assembly code in your C or C++ source code.

```
__asm [volatile] ("assembly instruction");
```

Example:

```
__asm volatile ("LDR R1, =0x11");
```

```
__asm volatile ("CPSID I"); //Mask interrupts
```

```
__asm volatile ("CPSIE I"); //Unmask interrupts
```

Thank you